Université de Yaoundé I
****
Faculté des Sciences
****
Département d'informatique
****

The University of Yaoundé I
****
Faculty of Sciences
****
Department of computer Science
****

Principes et Processus du Génie Logiciele: INF461 Travaux Pratiques
Session 2025/2026
Examinateur: Dr. Kimbi Xaveria

Objectif: Mise en place d'un Système de transactions bancaires multi-opérateurs

Contexte général

Vous développez un système de transactions bancaires utilisé par plusieurs opérateurs financiers (banques, opérateurs mobile money, etc.). Chaque opérateur a ses propres règles (taux, validations, workflows) mais doit s'exposer de façon uniforme au reste du système. Le système gère : **opérateurs, comptes, clients, administrateurs, transactions inter- et intra-opérateurs, notifications (via un service SMS externe) et plusieurs méthodes d'authentification**. L'accès aux tableaux de bord (client & admin) exige une authentification forte.

Votre mission est d'analyser, modéliser et implémenter une architecture logicielle robuste qui résout les différents besoins ci-dessous. Pour chaque besoin : proposer une **conception**, expliquer pourquoi elle résout le problème (diagramme UML minimal + justification), et livrer une **implémentation** (en Java/C#) + tests unitaires ou scénarios de test.

Objectif 1 : Le système doit supporter plusieurs méthodes d'authentification (mot de passe, empreinte, reconnaissance faciale, OTP). Le code qui orchestre l'authentification ne doit **pas** connaître à la compilation la méthode choisie : la sélection se fait à l'exécution (selon la configuration d'un opérateur et les préférences du client). concevoir un mécanisme de création/instanciation flexible pour ces méthodes (facilité d'ajout d'une nouvelle méthode sans modifier le code client). Fournir exemple d'extension (ajouter une méthode « Réalité augmentée ») et tests montrant la substitution dynamique.

Objectif 2: Chaque opérateur fournit un ensemble d'objets cohérents : validateurs de compte, calculateurs de taux, modules de notification, et adaptateurs vers des systèmes externes. L'ensemble d'objets d'un opérateur doit rester cohérent entre eux (par ex. un calculateur de taux X fonctionne avec le de façon centralisée. proposer une façon de regrouper et d'instancier ces familles d'objets pour un opérateur, et montrer comment basculer d'un opérateur A vers un opérateur B avec minimum de modifications.

Objectif 3: Une transaction (p.ex. transfert inter-opérateurs) est composée de plusieurs étapes optionnelles et paramétrables (vérifications, conversion de devise, application de commissions multiples, journalisation, notifications). Pour un certain type de transactions, l'ordre et la présence des étapes peuvent varier. Il faut un moyen propre de construire ces transactions pas à pas, et de

produire variantes (par exemple « transaction courte » vs « transaction complète ») sans dupliquer le code. concevoir une API fluide / explicite pour construire ces traitements transactionnels et montrer deux variantes de transactions construites avec ton API.

Objectif 4: Certaines informations doivent être accessibles globalement : instance de service de notifications (client du prestataire SMS), configuration d'authentification par défaut, et le gestionnaire d'événements système. Il faut garantir qu'il n'y ait **qu'une seule instance** de ces gestionnaires partagés pendant l'exécution et que l'accès soit thread-safe. concevoir une stratégie d'accès à ces ressources globales, montrant création, accès concurrent et nettoyage en fin d'exécution (test de concurrence minimal).

Objectif 5: Le prestataire SMS fournit une API REST différente selon le fournisseur (fournisseur A attend JSON X, fournisseur B SOAP, fournisseur C une URL particulière). Le reste de l'application veut appeler un service « envoyerNotificationSMS(dest, message) » de façon uniforme. On doit pouvoir changer de prestataire sans modifier le code métier. proposer une couche d'abstraction qui homogénéise l'usage du service SMS et montrer comment ajouter un nouveau fournisseur sans toucher le reste du système.

Objectif 6: Certains opérateurs partagent beaucoup de comportements communs (ex : gestion des comptes), tandis que d'autres diffèrent fortement pour un sous-ensemble d'opérations (ex : validation des transferts internationaux). L'architecture doit permettre de réutiliser un maximum de code commun modéliser la relation entre comportement commun et spécialisations d'opérateurs, et illustrer par 3 opérateurs différents (au moins un qui sur-définit une logique).

Objectif 7: Un administrateur peut envoyer : **une notification à un compte précis, à un groupe de comptes (par opérateur) ou à tous les clients du système (campagne globale).** Les notifications peuvent être composées (texte + pièce jointe + priorité). L'interface et le moteur d'envoi doivent pouvoir traiter une collection d'éléments simples et composés de façon identique (ex. envoyer une notification à un « ensemble » d'utilisateurs). proposer une structure de données pour représenter et parcourir ces cibles de notification (individuelles et groupées) et montrer l'algorithme d'envoi qui fonctionne quel que soit le niveau de composition.

Objectif 8: On souhaite permettre d'ajouter des comportements optionnels à un compte sans modifier la classe de compte de base (exemples : journalisation détaillée, verrouillage temporaire, plafonnement temporaire des retraits). Ces comportements doivent pouvoir être activés ou empilés à l'exécution pour un compte donné. implémenter au moins deux comportements « décoratifs » activables dynamiquement pour un compte et montrer leur empilement et retrait.

Objectif 9: Certains traitements suivent la même séquence générale (ex. validation d'ouverture de compte : vérifier documents → vérifier antécédents → finaliser + notifier), mais certaines étapes sont différentes selon l'opérateur. Il faut factoriser le squelette du processus tout en laissant des points d'extensions pour des détails spécifiques. modéliser au moins deux « squelettes » de workflow avec variantes opérateur-spécifiques (montrer où le comportement diffère).

Objectif 10: Il faut produire plusieurs types d'analyses/reportings sur les comptes et transactions : calculer le total des commissions perçues, vérifier fraudes potentielles, générer un rapport d'activité. Vous devez pouvoir ajouter de nouvelles opérations analytiques sans modifier les classes de comptes/transactions existantes, et pouvoir exécuter plusieurs analyses différentes en un seul

parcours des données (pour performance). concevoir une façon d'ajouter des opérations analytiques externes et d'appliquer une ou plusieurs d'entre elles sur les objets existants (compte / transaction) sans modifier leur implémentation.

Livrables demandés (pour chaque groupe de 5 Etudiants)

1. Diagramme UML simplifié (classes principales et relations) pour l'ensemble du système.
2. Pour **chaque** problème ($1 \rightarrow 10$) :
    - Description brève de la solution retenue + justification architecturale (pourquoi c'est adapté).
    - Un fragment d'implémentation (code réel ou pseudo-code) montrant l'essentiel.
    - Tests ou scénarios de démonstration (unitaires/minitests).
3. Rapport (max 6 pages) expliquant les compromis, difficultés rencontrées et possibilités d'évolution.
4. Cahier de charges, un document de conception et code source de l'application sur github.
5. Une interface graphique(React js, ou Angular, flutter) est fortement recommandée.

Jour de la présentation finale: **12 Janvier 2026**